TURING 图灵程序设计丛书

真实项目中的经验总结

行业精英们的智慧结晶



# Python 开发实战

团队开发环境的搭建与管理 / ticket驱动开发 / 源代码管理（Mercurial）/ PyPI包
Jenkins持续集成 / 性能优化（nginx与gunicorn）/ GAE开发 / Django框架……

【日】BePROUD股份有限公司 著
盛 荣 译 张靖强 审

人民邮电出版社
POSTS & TELECOM PRESS



# Python
# 开发实战

## Python Professional Programming

本书来自真正的开发现场，是BePROUD公司众多极客在真实项目中的经验总结。

作者从Python的环境搭建开始讲起，介绍了Web应用开发方法、任务管理，以及评审、测试及高效部署、服务器调试等内容，尽可能网罗了Python开发流程中的方方面面。在这里，Python仅仅是一个载体，很多知识点在非Python下也适用，这也是本书最大的特色所在。

本书适合有一定基础的Python开发者，以及使用PHP或Ruby进行网页开发的读者阅读。






图灵社区：iTuring.cn
热线：(010)51095186转600

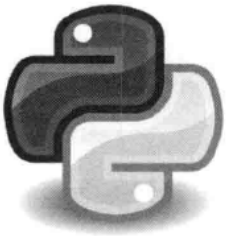
分类建议 计算机/程序设计

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-32089-6
定价：79.00元

TURING 图灵程序设计丛书



# Python 开发实战

【日】BePROUD股份有限公司 著
盛 荣 译 张靖强 审

人民邮电出版社
北 京

**图书在版编目（CIP）数据**

Python开发实战 / 日本BePROUD股份有限公司著 ；盛荣译. -- 北京 ：人民邮电出版社，2014.6
（图灵程序设计丛书）
ISBN 978-7-115-32089-6

Ⅰ. ①P… Ⅱ. ①日… ②盛… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.56

中国版本图书馆CIP数据核字(2013)第115494号

## 内 容 提 要

本书来自真正的开发现场，是 BePROUD 公司众多极客在真实项目中的经验总结。

作者从 Python 的环境搭建讲起，介绍了 Web 应用开发方法、任务管理，以及评审、测试及高效部署、服务器调试等内容，尽可能网罗了 Python 开发流程中所涉及的方方面面。在这里，Python 仅仅是一个载体，很多知识点在非 Python 下也适用，这也是本书最大的特色所在。

本书适合有一定基础的 Python 开发者，以及使用 PHP 或 Ruby 进行网页开发的读者阅读。

◆ 著　[日] BePROUD股份有限公司
译　盛 荣
审　张靖强
责任编辑　乐 馨
执行编辑　金松月
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京天宇星印刷厂印刷

◆ 开本：800×1000　1/16
印张：24.5
字数：379千字　2014年 6 月第 1 版
印数：1－4 000册　2014年 6 月北京第 1 次印刷
著作权合同登记号　图字：01-2012-9107号

定价：79.00元

**读者服务热线：(010)51095186转600　印装质量热线：(010)81055316**
**反盗版热线：(010)81055315**
**广告经营许可证：京崇工商广字第 0021 号**

# 版权声明

*PYTHON PROFESSIONAL PROGRAMMING*
Copyright © 2012 by BeProud
All rights reserved.
Originally published in Japan by Shuwa System CO.,LTD. Tokyo.
Chinese (in simplified character only) translation rights arranged with Shuwa System CO., LTD. through CREEK & RIVER Co.,LTD. Tokyo

本书中文简体字版由Shuwa System Co., LTD.授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。
版权所有，侵权必究。

# 前　言

BePROUD 股份有限公司在众多的项目开发中使用了Python编程语言。我们希望通过撰写本书，与你分享在项目实践中所积累Python使用技巧。

最近，公司的员工数量不断增加，为了让新员工快速上手，我们亦将在BePROUD工作必须掌握的基础知识写入本书中。

本书从整个工程环境的建立开始，介绍了Web应用开发方法、任务管理，以及评审、测试及高效部署、服务器调试等内容，尽可能网罗了Python开发流程中所涉及的一系列内容。如书名所示，本书更多关注项目实战技巧。

在进入正题前，我想先谈谈BePROUD公司日常工作的思维方式。

## ◉Geek/Nerd[①]云集的公司

BePROUD是一个Geek和Nerd众多的公司，云集了许多行业精英。

出于兴趣，这些员工经常利用自己的业余时间学习，将学到的知识付诸实践。作为Geek/Nerd，他们永远不会吝惜时间和精力。

当然，Geek/Nerd中有许多怪人，但BePROUD公司所有成员都有这样的共识：

- 尽量简化不想做的工作。
- 掌握了好的方法就要付诸实践。
- 带着好心情快乐地工作。

## ◉尽量简化不想做的工作

在工作中，重复地做一项工作是非常无聊的。我们都尽量想办法使其一劳永逸。另外，我们也不喜欢那些过程复杂且又容易失败的工作。因此我们尽可能将其简化，防止失败。

## ◉掌握了好的方法就要付诸实践

世间总有一些好办法或一些新想法，我们总是积极地在工作中去尝试。

在工作中去实践好方法往往可以减轻我们的负担。但在实践之前，我们必须先甄别出可以真正起到作用的方式方法，然后再开始学习、应用。

## ◉带着好心情快乐地工作

如前所述，当掌握了好的方法，提高工作效率之后，我们就可以抱有一种好心情去享受工作

① Geek和Nerd都有痴迷某物的意思，本书指的是对于软件开发等领域极度痴迷的人。——译者注

了。比如做一个好玩的Skype机器人，在下班后找一个空闲的会议室玩LT游戏等。我们都认为，公司不仅是工作场所，也是一个让大家在做好工作的同时又能够享受它的地方。

本书的内容基于BePROUD公司员工的实践经验，因此不会强迫你去单纯地记忆复杂的知识，而是为你提供能够在实践中直接应用的内容。如果本书中介绍的知识能够对你的实际工作有帮助，我们将感到莫大的欣慰。

## ◉谢辞

感谢以下IT界同行审阅本书：寺田学（@terapyon）、铃木孝典（@takanory）、田村佑树（@isoparametric）、川口耕介（@kohsukekawa）、和田贵久（@wadatka）、矢野勉（@t_yano）、涉川芳树（@shibukawa）、西尾泰和（@nishio）、饭野卓见（@troter）、关根佳直、四柳刚（@ytyng）、波田野欲一（@tcsh）。大家在百忙之中，非常爽快地接受了审稿工作并且给出了中肯的意见，再次深表感谢。还有BePROUD公司的小田切笃（@aodag）、菊田刚（@nyomui）、高桥和之（@kazyk）等，他们在繁忙的项目中抽出时间来审阅书稿，同样非常感谢他们。

最后，还要感谢那些为本书执笔却没有署名的BePROUD的各位员工。没有大家平时的斟酌切磋，本书也是无法完成的。

正是得到大家的鼎力相助，本书才得以完成。让我们一起祝愿本书能够为IT业界奉献微薄之力。

本书所有执笔者

# 导　读

## ◉本书涵盖的内容

本书分成四部分，共15章。

第一部分以个人开发为重点，介绍了如何在个人开发中构建Python开发环境（第1章），并说明了开发简单的Python应用程序的方法（第2章）。

第二部分介绍了团队开发的相关内容，依次探讨了如何准备团队开发环境（第3章）、文档（第4章）、项目管理与评审（第5章）、模块设计与单元测试（第6章）、库的发布与打包（第7章）、版本控制（第8章）、持续集成（第9章）等。这些主题都是在提高团队开发效率方面必不可少的技术与方法。

第三部分介绍了Web服务发布的环境构建与运用所必须的技术（第10章）和性能调优的方法（第11章），并详细探讨了使用Google App Engine进行Web应用构建的方法（第12章）。

第四部分先阐述了将测试引入整个开发流程从而提高项目效率这一话题（第13章），随后从Django的基础开始讲起，介绍它在实践中的使用方法（第14章）。最后讲解了提高开发速度的诀窍，如Python中实用的开发模块（第15章）。

## ◉阅读本书时需要准备

### 软件环境以及版本

- 操作系统：Ubuntu-11.10
- Python：2.7.2

### 关于操作系统

主机可以使用Windows/Mac OS/Linux等作为服务器的测试环境，运行虚拟机，在虚拟机上运行Ubuntu。

### Python的官方网站帮助链接

http://www.python.jp/doc/2.7/[①]

Python官方网站的参考文档一般只介绍最基本的使用方法，有时会省略部分说明，因此建议你常备更详细的参考手册慢慢阅读。

Python官方网站的学习教程非常适合用来学习Python的安装步骤、基本语法、类以及模块。该教程中所涉及的知识，本书将不再赘述。

---

① 可以参阅http://docs.python.org/release/2.7/ 。——译者注

#### Unix/Linux的一般操作命令

本书采用Ubuntu Linux作为操作系统进行阐述，因此有关Unix/Linux的基本操作均不再介绍。你如果想知道相关的操作命令，可以参考http://linuxjm.sourceforge.jp/。

#### 关于PyPI（Python Package Index）

PyPI（http://pypi.python.org/pypi）提供了众多使用pip工具进行安装的Python程序包，并且该工具还提供了程序包自动安装的相关功能[①]。本书用到的程序包均可在PyPI站点中找到。

#### 关于敏捷开发和极限编程

本书不会针对敏捷开发[②]和极限编程[③]作个别说明。关于这两个主题有众多的书籍和网站可以供参考。

## ◉本书读者对象

本书适合如下读者：

- 希望改善个人开发环境的开发者
- 希望使团队开发更上一层楼的开发者
- 在工作中使用Python环境并想学习相关技术的人
- 新加入BePROUD公司项目的人

① 非常类似Perl的CPAN。——译者注
② 即agile。——译者注
③ 即eXtreme Programming 。——译者注

# 目　录

## 第一部分　使用 Python 开发


01　开始使用 Python……2
01.01　Python 环境搭建……2
01.01.01　安装 apt 软件包……2
01.01.02　easy_install 与 pip……3
01.01.03　安装 virtualenv……4
01.01.04　使用不同版本的 Python……9
01.02　Mercurial 环境搭建……12
01.02.01　Mercurial 的简介……12
01.02.02　Mercurial 的安装……12
01.02.03　代码库的建立……13
01.02.04　文件的操作……14
01.03　编辑器与方便的开发工具……17
01.03.01　编辑器……17
01.03.02　Python 开发小贴士……19
01.04　本章小结……22

02　开发 Web 应用程序……23
02.01　了解 Web 应用程序……23
02.01.01　什么是 Web 应用程序……23
02.01.02　Web 应用程序与桌面应用程序的不同……24
02.01.03　Web 应用程序的机制……24
02.02　Web 应用程序开发的事前准备……27
02.02.01　关于 Flask……27
02.02.02　Flask 的安装……27
02.03　Web 应用程序的开发流程……28
02.04　明确需要开发的目标系统……28
02.04.01　留言簿应用程序的需求……28
02.04.02　明确必要的功能……29
02.04.03　明确页面的表现形式……29
02.05　页面设计……29
02.05.01　页面设计草图……30
02.05.02　设计 HTML 和 CSS……30
02.06　后台功能的开发……35
02.06.01　保存评论数据……35
02.06.02　取出保存过的评论数据……36
02.06.03　使用模板引擎显示页面……37
02.06.04　准备评论的目标 URL 地址……39
02.06.05　调整模板输出……40
02.07　功能验证……41
02.08　本章小结……42


## 第二部分　团队开发的周期和流程


03　准备团队开发环境……46
03.01　在公共服务器上建立用户、设置权限……46
03.01.01　用户与用户组的建立……46
03.01.02　sudoers……46
03.01.03　virtualenv……47
03.02　问题跟踪系统……47
03.02.01　Trac……48
03.02.02　安装 Trac……48
03.03　版本控制系统……51
03.04　有利于团队开发的工具……52
03.04.01　Skype……52
03.04.02　DropBox……53
03.04.03　Google Docs……54
03.05　本章小结……54

04　编写开发文档的准备……55
04.01　编写项目开发文档……55
04.01.01　在写文档时，我们不想做什么工作……55

04.01.02 愿意在什么样的条件下编写文档……57
04.02 Sphinx 的基础与安装……58
04.02.01 Sphinx 的安装……58
04.02.02 reStructuredText 入门……60
04.02.03 使用 Sphinx 编写结构化文档的流程……61
04.02.04 Sphinx 的扩展……66
04.03 引入 Sphinx 后解决的问题与新的问题……67
04.03.01 采用纯文本，可使用一般的编辑器来写文档……67
04.03.02 内容与形式分离，不关注文档形式而专注于内容的编写……68
04.03.03 不仅仅是 PDF，通过一个源可以输出多种格式……71
04.03.04 将文档分割成多个文件编辑，进行结构化处理……72
04.03.05 通过 Mercurial 进行简单的版本控制……73
04.03.06 将 API 参考手册和对应程序协同管理……73
04.03.07 一般文档可通过 Web 浏览器共享……76
04.03.08 Sphinx 引入后仍需探讨的问题……76
04.04 文档集合的创建与利用……77
04.04.01 什么是文档集合……78
04.04.02 项目中必不可少的文档列表……78
04.04.03 面向团队领导、经理……79
04.04.04 面向设计人员……80
04.04.05 面向开发人员……80
04.04.06 面向用户……81
04.05 本章小结……81
05 问题跟踪与评审……82
05.01 任务管理与 ticket 驱动开发……82
05.01.01 开始写代码……在这之前还是先创建一个 ticket……82
05.01.02 创建和 ticket 编号一致的分支……83
05.01.03 让版本发布与分支对应……84
05.01.04 分支的合并……84
05.01.05 整理 ticket 信息……86
05.01.06 ticket 的分割……86
05.02 评审……87
05.02.01 为什么需要评审……87
05.02.02 作为被评审人：代码评审篇……87
05.02.03 作为被评审人：工作评审篇……88
05.02.04 作为评审人：代码评审篇……89
05.02.05 作为评审人：工作评审篇……92
05.03 本章小结……92
06 模块的分割设计与单元测试……93
06.01 模块分割设计方法……93
06.01.01 功能设计……93
06.01.02 Web 应用程序的构成组件……94
06.01.03 组件设计……96
06.01.04 模块与包……97
06.02 测试……99
06.02.01 测试的种类……99
06.02.02 编写单元测试……101
06.02.03 从单元测试中去除环境依赖……109
06.02.04 使用 WebTest 进行功能测试……114
06.03 从测试来改善设计……118
06.04 迈向测试执行的自动化……120
06.04.01 测试环境的自动生成……120
06.04.02 可以反复执行的测试环境……121
06.05 本章小结……121

07 打包与自动建立环境……122
07.01 源代码打包与发布自动化……123
07.01.01 打包……123
07.01.02 程序包的自动化发布……127
07.01.03 向 PyPI 注册程序包……128
07.01.04 向 PyPI 上传程序包……128
07.01.05 书写程序包的使用说明…129
07.02 安装与环境关联库的介绍……131
07.02.01 Distribute……132
07.02.02 pip……133
07.02.03 virtualenv……134
07.03 应用环境自动构建所必备的功能与工具……135
07.03.01 准备独立的 Python 环境……135
07.03.02 Python 库的程序包化以及标准安装……136
07.03.03 锁定应用环境的程序包版本……136
07.03.04 整个环境可以离线安装…136
07.03.05 重建环境时可以复用缓存……137
07.03.06 使用编译好的二进制模块……138
07.03.07 当 PyPI 宕机或高峰时，有备用服务器……138
07.04 自动应用环境构建……140
07.04.01 开始……140
07.04.02 构建初始环境……140
07.04.03 包含源代码的应用环境……141
07.04.04 在部署环境中的离线安装……143
07.04.05 安装部署需要二进制编译的程序包……144
07.05 公司内部共享专用库……144
07.05.01 在多个项目间共享代码的方法……145
07.05.02 自动建立非公开程序包的应用环境……149
07.06 本章小结……152
08 使用 Mercurial 管理源代码……153
08.01 Mercurial 的代码库管理与配置……153
08.01.01 在服务器上设置 UNIX 用户与用户组……153
08.01.02 设置 umask 值……154
08.01.03 创建代码库……154
08.01.04 配置 hgrc……155
08.01.05 使用配置后的代码库……155
08.01.06 使用 hgweb 作为简易的中央代码库……155
08.02 灵活使用钩子功能……156
08.02.01 钩子功能的设置方法……157
08.02.02 使用钩子脚本……157
08.02.03 钩子事件……157
08.02.04 钩子执行的时机……159
08.02.05 编写钩子脚本……161
08.03 分支操作……164
08.04 分支间的合并……165
08.04.01 没有冲突的合并……165
08.04.02 用文本编辑器消除合并冲突……167
08.04.03 合并的模式与冲突的种类……169
08.04.04 使用图形界面合并的方法……170
08.05 GUI 客户端……173
08.05.01 GUI 客户端的介绍……173
08.05.02 GUI 客户端的优点……176
08.05.03 GUI 客户端的缺点……179
08.05.04 hgwebcommit……180
08.06 考虑实际运用而产生的 BP Mercurial W/F……182
08.06.01 概要……182
08.06.02 背景……183
08.06.03 代码库的构成……183
08.06.04 源代码提交……184
08.06.05 设计师的提交操作……186
08.06.06 分支的合并……187

08.06.07 集成分支……188
08.07 本章小结……189
09 使用 Jenkins 持续集成……192
09.01 什么是持续集成……192
09.01.01 关于持续集成……192
09.01.02 关于 Jenkins……194
09.02 Jenkins 的安装……194
09.02.01 安装 Java……194
09.02.02 安装 Jenkins 主体程序……194
09.02.03 本章用到的 Jenkins 插件……195
09.03 执行测试代码……195
09.03.01 简单测试代码的示例……195
09.03.02 注册作业……196
09.03.03 作业执行成功与失败……198
09.04 测试结果通过报表输出……199
09.04.01 安装 nose……199
09.04.02 调用 nosetests 命令……199
09.04.03 为使用 nose 修改 Jenkins 配置……200
09.05 展示代码覆盖率报告……201
09.05.01 coverage 的安装……201
09.05.02 coverage 命令的概述：分为“统计”和“输出”两步……201
09.05.03 通过 nose 执行 coverage…201
09.05.04 读取代码覆盖率的报告…202
09.06 执行 Django 测试……204
09.06.01 Django 模块的安装……204
09.06.02 Django 的安装……204
09.06.03 样本代码……205
09.06.04 Jenkins 的创建……208
09.06.05 “构建后处理”部分的配置……210
09.07 通过 Jenkins 构建文档……212
09.07.01 Sphinx 的安装……212
09.07.02 配置 Jenkins 作业的注册信息……212
09.07.03 Sphinx 的构建在出现告警时作业失败……213
09.07.04 查阅编译成果……214
09.07.05 使用 Task Scanner Plugin 管理 TODO 信息……215
09.07.06 Task Scanner Plugin 的配置示例……216
09.08 进一步灵活使用 Jenkins……216
09.08.01 便捷的功能……216
09.08.02 进一步的改善……218
09.09 本章小结……219

## 第三部分 服务的公开

10 自动构建和部署环境……222
10.01 安装与配置具有依赖关系的程序包……222
10.01.01 创建操作用户……223
10.01.02 列举必要的程序包……224
10.01.03 封闭环境的安装……226
10.01.04 应用程序的配置……228
10.01.05 环境验证……231
10.02 整理部署的操作步骤……231
10.02.01 赋予代码库访问权限……232
10.02.02 克隆源代码库……232
10.02.03 部署方法总结……232
10.03 使用 Fabric 将操作步骤自动化……233
10.03.01 什么是 Fabric……233
10.03.02 安装与环境配置……233
10.03.03 编写简单的脚本……234
10.03.04 参考步骤说明书进行自动化……234
10.03.05 验证执行行为……238
10.03.06 添加注释……238
10.04 本章小结……239
11 改善应用程序的性能……240
11.01 Web 应用程序的性能……240

11.01.01　Web 应用遭遇大量请求时会如何 ……240
11.01.02　高负载时的对策 ……241
11.02　留言簿应用程序的性能测试 ……242
11.02.01　什么是应用程序的性能 …242
11.02.02　ApacheBench 的安装 ……242
11.02.03　使用 ApacheBench 检测性能 ……242
11.03　关于 gunicorn ……245
11.03.01　gunicorn 的安装 ……245
11.03.02　使用 gunicorn 运行应用程序 ……245
11.04　关于 nginx ……247
11.04.01　nginx 的安装 ……248
11.04.02　nginx 的性能测试 ……248
11.05　使用 nginx 和 gunicorn 共同承载应用程序 ……250
11.05.01　gunicorn 的配置 ……250
11.05.02　nginx 的配置 ……251
11.05.03　测定 nginx+gunicorn 组合的性能 ……251
11.05.04　性能的比较 ……252
11.06　本章小结 ……253
12　Google App Engine ……254
12.01　关于 App Engine ……254
12.01.01　什么是 App Engine ……254
12.01.02　App Engine 的优点 ……255
12.01.03　App Engine 平台的局限性 ……255
12.01.04　使用案例 ……256
12.02　App Engine 的主要功能 ……256
12.02.01　数据存储 ……256
12.02.02　自动扩展与负载分散 ……257
12.02.03　在 App Engine 中的 Python 运行环境 ……259
12.03　App Engine 的开发准备工作与步骤确认 ……260
12.03.01　获取账号并开发程序 ……260
12.03.02　创建 SDK ……263
12.03.03　开发简单的应用程序 ……264
12.03.04　部署 ……266
12.04　在 App Engine 中开发 Flask 的应用程序 ……267
12.04.01　使已有的应用程序运行 …267
12.04.02　部署并运行 ……271
12.04.03　管理终端 ……272
12.05　调整：改善应用程序性能 ……274
12.05.01　根据 Appstats 进行性能测试 ……274
12.05.02　使用多线程来处理 ……276
12.06　性能调优：减少计费额 ……277
12.06.01　与计费有关的配置 ……277
12.06.02　计费表 ……277
12.06.03　节约实例的数量 ……279
12.06.04　减少数据存储的操作 ……280
12.07　测试 ……282
12.07.01　在测试代码内使用 API …282
12.07.02　测试的执行 ……284
12.07.03　在真实服务器上测试 ……285
12.08　本章小结 ……285

## 第四部分　加速开发的技术

13　测试是不可分割的一部分 ……288
13.01　认清测试现状 ……288
13.02　在开发的各个阶段引入测试工作 ……289
13.02.01　文档的测试（文档评审） ……289
13.02.02　编写测试的方法（输入与输出） ……292
13.02.03　测试的执行与测试阶段的轮换（把什么工作做到什么程度） ……295
13.03　本章小结：对测试不要抱有恐惧 ……298
14　便捷地使用 Django ……300
14.01　什么是 Django ……300

14.01.01 Django 的安装 …… 300
14.01.02 Django 的架构 …… 300
14.01.03 Django 的文档 …… 304
14.02 让数据库的集成更加方便 …… 304
14.02.01 什么是数据库的集成 …… 304
14.02.02 默认情况下 Django 能做什么 …… 304
14.02.03 对，就是它，使用 South 就能够做到 …… 305
14.03 使用 fixture replacement 使测试更加方便 …… 312
14.03.01 什么是测试配置器（fixture） …… 312
14.03.02 默认测试配置器的不便之处 …… 314
14.03.03 使用 factory_boy …… 315
14.04 使调试更加方便 …… 318
14.05 本章小结 …… 324
15 使用方便的 Python 模块 …… 325
15.01 简化日期计算的模块 …… 325
15.01.01 日期计算的复杂之处 …… 325
15.01.02 引入 dateutil …… 327
15.02 简化模型映射的模块 …… 329
15.02.01 必须进行模型映射的理由 …… 329
15.02.02 映射规则的构造与复用 …… 330
15.02.03 导入 bpmappers 模块 …… 333
15.02.04 与 Django 的联动 …… 335
15.02.05 编写 JSON API …… 336
15.03 判断字符的编码 …… 337
15.04 RSS 阅读订阅的解析模块 …… 339
15.04.01 导入 feedparser 模块 …… 339
15.04.02 使用 feedparser 解析 RSS 阅读订阅信息 …… 339
15.05 图像处理模块 …… 340
15.05.01 安装 PIL 模块 …… 340
15.05.02 转换图像格式 …… 341
15.05.03 调整图像的尺寸 …… 342
15.05.04 截取图像的某个部分 …… 344
15.05.05 图像的过滤处理 …… 345
15.06 数据的加密处理模块 …… 346
15.06.01 PyCrypto 的安装 …… 346
15.06.02 共同密钥的加密与解密 …… 347
15.06.03 公开密钥方式的加密与解密 …… 347
15.07 调用 Twitter 的 API 模块 …… 351
15.07.01 导入 tweepy 模块 …… 351
15.07.02 应用程序的登录域和 Consumer Key 的获取 …… 351
15.07.03 获得 Access Token …… 354
15.07.04 调用 TwitterAPI …… 354
15.07.05 开发使用 Twitter 认证的应用程序 …… 356
15.08 本章小结 …… 361
附录 A 建立 VirtualBox 环境 …… 362
附录 B 建立操作系统环境 …… 366

# Part 1 第一部分

# 使用 Python 开发

各位读者，当你在学习一门新技术或者新的编程语言时，是不是常有无从下手的感觉？初学者在学习前，往往不知道事先应该准备什么；或刚开始准备，就马上在安装问题上栽了跟头；抑或是准备工作完成后，打算开始学习时，才发现事先应该安装的软件没有安装到位等。上述各种情况最终导致初学者在学习之初，就花费了大量的时间。

在构建 Python 的开发环境时，往往会遇到不同操作系统版本和 Python 版本，我们需要有一个兼容平滑的开发环境。此时，个人开发者尤其是自学者几乎都是参考书本或者 Web 站点等各种信息，并根据自己的需要建立开发环境。虽然这些开发环境并不完全相同，但有许多共同点。

在本书第 1 章中，我们将个人开发者建立开发环境的步骤尽可能深入浅出地展现出来。即使是初学者，也可按照里面的步骤进行操作。随后在第 2 章中，我们将会尝试实际开发一个 Web 应用。

本部分内容


- 第 1 章　开始使用 Python
- 第 2 章　开发 Web 应用程序

# 01 开始使用Python

本章我们将学习以下内容。我们的说明会在虚拟机环境上进行，将用来学习的主机称为“Host系统”，并将虚拟机上安装的系统称为“Guest系统”。如果你自己已经准备好了Python的开发环境，跳过本章也没有关系。

- Python 环境搭建
- Mercurial 环境搭建
- 编辑器与方便的开发工具

## 01.01 Python环境搭建

本节介绍在Ubuntu下开发Python时，需要安装的一些实用的工具包。

NOTE

本书默认的操作系统为Ubuntu 11.10 （Server版）。该操作系统运行在Oracle VirtualBox虚拟机上。如果你还没有掌握这类环境的搭建方法，可以参考本书的附录A。初学者也请先参照附录，再阅读下面的内容。

### 01.01.01 安装apt软件包

Ubuntu使用aptitude命令进行操作系统的软件包管理[①]。首先我们使用下面的命令更新所有的安装包，这些安装包中包括了Python开发所必备的工具[②]。

apt软件包的更新与升级

```
$ sudo aptitude -y update
$ sudo aptitude -y upgrade
$ sudo aptitude -y install build-essential
$ sudo aptitude -y install libsqlite3-dev
$ sudo aptitude -y install libreadline6-dev
$ sudo aptitude -y install libgdbm-dev
$ sudo aptitude -y install zlib1g-dev
$ sudo aptitude -y install libbz2-dev
$ sudo aptitude -y install sqlite3
$ sudo aptitude -y install tk-dev
$ sudo aptitude -y install zip
```

① aptitude类似Redhat的rpm，属于Linux“方言命令”。——译者注

② 运行命令前须先联网。——译者注

在命令中，-y的作用是告知安装程序：在安装过程中，如果遇到Y或者N的提问，一律以Yes作为默认答案。如果你需要在安装过程中逐个确认，可以将-y去掉，再执行该命令。

build-essential 能够在Ubuntu上编译安装Python所必备的一系列工具链（如gcc、make等）。这些基本的工具链，在Python以及接下来提及的模块和包中，均会作为必要的使用工具，在这里就先行一步一起安装。[①]

接下来我们安装Python相关的包。

**Python相关包的安装**

```
# python-dev包的安装
$ sudo aptitude -y install python-dev

# Distribute包的安装
$ sudo chmod -R 0775 /usr/local
$ sudo chgrp –R bpbook /usr/local
$ wget http://python-distribute.org/distribute_setup.py
$ sudo python distribute_setup.py
```

Distribute 是支持Python模块构建与导入的工具包，虽然使用apt也能够安装，但通常安装的版本较老，不如将源代码下载下来，进行编译安装较为妥当。关于Distribute包的详细信息可以参考本书第7章。

Python相关包的安装到这里就结束了，最后我们从Ubuntu上确认一下默认安装的Python版本。

**Python版本的确认**

```
$ python -V
Python 2.7.2+
```

输入上面的命令，就可以知道本机现在所安装的Python版本，在Ubuntu 11.10上所安装的版本是Python 2.7.2 +。

### 01.01.02 easy_install 与pip

Python有一个叫做PyPI（Python Package Index，http://pypi.python.org/pypi）的公共资源库，它管理着Python相关的各个功能包[②]。任何人都可以登录该主页去下载Python相关的包资源。在使用Python开发某些软件时，我们经常会从PyPI下载安装一些必要的功能包。

**NOTE**

PyPI的发音为/P I Pi/。

可以想象，同Python和PyPI相类似的组合有这样几个：Perl和CPAN（http://www.cpan.org/）、Ruby和RubyGems（http://rubygems.org/）、PHP和PEAR（http://pear.php.net/）。

① 该命令表示更改/usr/local的目录所属用户组，读者可根据自己情况修改。——译者注

② 熟悉Perl的读者可以将PyPI理解为Perl中的CPAN。——译者注

从PyPI安装功能包和模块时所使用的命令有：easy_install和pip。下面我们以pip为例进行介绍。

pip命令的安装

```
$ wget https://raw.github.com/pypa/pip/master/contrib/get-pip.py
$ sudo python get-pip.py
```

在安装完成后，我们来检查一下pip的版本。

pip版本的确认

```
$ pip --version
pip 1.0.2 from /usr/local/lib/python2.7/dist-packages/pip-1.0.2-py2.7.egg
(python 2.7)
```

在作者执笔时，安装的pip版本是pip 1.0.2。在此之后，本书介绍的所有Python相关模块的安装，均以pip命令的形式进行说明。使用pip命令安装的包会存放在/usr/local/python2.7目录下。

NOTE

在pip编译及安装的过程中，需要使用gcc等编译构建工具。本书默认的是Ubuntu，gcc已经在build-essential中进行了安装。在Windows操作系统中，我们同样需要事先安装Visual Studio 2008免费版本或者MingGW等开源软件，这些都是不收费的。

另外，PyPI还提供了已经编译好的二进制文件egg给用户直接使用，如果用户直接使用egg的话，就无需在操作系统中准备编译器了。egg是以平台相关的形式发布的，多数面向Windows用户。

pip命令没有使用和egg一样的发布形式，如果你想使用egg安装的话，可以使用easy_install命令。有关easy_install和pip的内容我们将在第7章深入地学习。

## 01.01.03　安装 virtualenv

这里的virtualenv是将Python的实际运行环境进行虚拟的工具包。

使用pip install命令安装的功能包，均会存放在/usr/local/lib/python2.7目录下。一旦安装了各种各样的功能包，当我们想让不同的Python应用程序在同一个环境开发时，就会发现自己根本分不清哪个包对应哪个应用程序。virtualenv就是在这样的情况下，对多种不同的Python运行环境进行管理的一个工具。

安装virtualenv

```
$ pip install virtualenv
```

安装完成后，使用帮助命令查看一下。

安装virtualenv之后，使用help命令

```
$ virtualenv --help
```

### ◉ virtualenv的使用方法

接下来将简要介绍一下virtualenv的使用方法。首先用pip提供的pip freeze确认已经安装好的所有包的版本信息。

使用pip freeze确认

```
$ pip freeze
GnuPGInterface==0.3.2
Landscape-Client==11.07.1.1
PAM==0.4.2
Twisted-Core==11.0.0
apt-xapian-index==0.44
chardet==2.0.1
command-not-found==0.2.44
distribute==0.6.24
httplib2==0.7.1
keyring==0.6.2
launchpadlib==1.9.8
lazr.restfulclient==0.11.2
lazr.uri==1.0.2
oauth==1.0.1
pexpect==2.3
pyOpenSSL==0.12
pycrypto==2.3
pycurl==7.19.0
.
.
.
```

当输入命令后，就可以看到所有包的清单，这些就是已经安装在/usr/local/lib/python2.7下的所有包的信息。

接下来，用virtualenv创建出一个供新开发的Python应用程序运行的虚拟环境。先在home目录下建立一个工作目录，在该工作目录中建立一个虚拟的运行环境，下面的命令将完成这些工作。

使用virtualenv建立虚拟运行环境

```
$ export VIRTUALENV_USE_DISTRIBUTE=true
$ mkdir ~/work
$ cd ~/work
$ virtualenv env
```

上述命令执行后，会在work目录下建立一个新的env目录，这就是虚拟环境目录。

接下来我们简单说明一下与virtualenv相关的环境变量VIRTUALENV_USE_DISTRIBUTE。

VIRTUALENV_USE_DISTRIBUTE是向Distribute明示Python运行信息的环境变量[①]。如果在开发过程中，希望所开发的Python应用程序有“不兼容旧版本的Python”

① 即发布Python应用时，告知虚拟环境中相关的运行信息。——译者注

或者“兼容最新版本Python”的要求，最好对该环境变量进行设定。为了不在每次登录都输入这样的命令，我们可以在登录的shell配置中加入下面这条语句。

```
$ export VIRTUALENV_USE_DISTRIBUTE=true
```

Distribute会根据这个环境变量是否存在，判断虚拟运行环境的配置有效或无效。因此在不适用Distribute的时候，建议用unset命令取消该环境变量。

**NOTE**

virtualenv的默认设置是创建虚拟运行环境时不使用/usr/local/lib/python2.7/site-packages下的模块或者包。相反，当我们需要使用系统下的模块或者包来创建虚拟运行环境的话，可以使用--system-site-packages选项：

```
$ virtualenv --system-site-packages env
```

--system-site-packages是在virtualenv版本1.7之后添加的，包括1.6在内的之前版本都默认使用系统下的模块或者包来创建虚拟运行环境。因此，如果在1.6以前的版本下建立最小虚拟运行环境，可以使用--no-site-packages选项。

```
$ virtualenv --no-site-packages env
```

接下来，激活虚拟运行环境。

**虚拟运行环境激活**

```
$ source env/bin/activate
```

当使用了source命令后，虚拟运行环境信息开始生效。我们可以通过查看命令行终端开头是否出现了（env）这样的标志，来判断虚拟运行环境是否彻底被激活。如图1-1所示。

**图1-1：virtualenv被激活的状态**

```
(env)                          :~/work$
```

当虚拟运行环境被激活后，我们可以再一次使用pip freeze命令来检查安装好的包版本信息。

**检查安装好的包版本信息**

```
(env) $ pip freeze
distribute==0.6.24
wsgiref==0.1.2
```

可见，我们已经建立了一个安装最少数量包的Python虚拟运行环境。在这样的环境下，可以再一次使用pip install命令安装开发应用程序需要的包或者模块。如此一来，我们就可以很容易地把握在开发Python应用程序时需要安装的包和模块，从而尽可能减少其他不相干包或者模块给应用程序开发带来的未知影响。

如果需要退出当前的虚拟运行环境，可以使用下面的命令。

**退出虚拟运行环境**

```
(env) $ deactivate
```

当使用virtualenv建立虚拟运行环境来开发应用程序时，千万不要忘了先激活虚拟运行环境。当不再需要该虚拟环境时（这里就是指名为env的虚拟环境），可以直接使用rm等命令将对应的目录删除。

### ◉ virtualenvwrapper

接着我们介绍一种比virtualenv更容易使用的工具——virtualenvwrapper。virtualenv需要每次使用source命令导入虚拟运行环境信息，这一点非常麻烦，另外开发者还有可能忘记虚拟环境目录的建立位置。virtualenwrappper这一命令行工具就是通过对virtualenv进行二次封装，解决了上述种种问题。让我们先退出虚拟运行环境，使用pip命令安装virtualenvwrapper。

**virtualenvwrapper的安装**

```
$ pip install virtualenvwrapper
```

安装完成后，会在下面的位置生成virtualenvwrapper的shell脚本。

/usr/local/bin/virtualenvwrapper.sh

在使用virtualenvwrapper时，需要配置登录的shell初始化脚本，将virtualenvwrapper.sh的信息读入当前的shell环境。这里以bash为例，通过对用户目录下的.bashrc配置文件进行如下修改即可。[①]

**修改.bashrc**

```
if [ -f /usr/local/bin/virtualenvwrapper.sh ]; then
    export WORKON_HOME=$HOME/.virtualenvs
    source /usr/local/bin/virtualenvwrapper.sh
fi
```

准备工作到此结束。由于.bashrc是在登录时执行的，因此如果登出了虚拟机上客户端操作系统，再一次登录之后，运行下面的命令同样可以使用virtualenvwrapper命令。

**再次读入.bashrc**

```
$ source ~/.bashrc
```

我们可以输入命令mkvirtualenv，看一下其是否可用。

① 这里修改登录shell配置是为了让用户每次登录以后都生效。——译者注

确认mkvirtualenv的帮助信息

```
$ mkvirtualenv --help

Usage: virtualenv [OPTIONS] DEST_DIR

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v, --verbose         Increase verbosity
  -q, --quiet           Decrease verbosity
  -p PYTHON_EXE, --python=PYTHON_EXE
                        The Python interpreter to use, e.g.,
                        --python=python2.5 will use the python2.5
                          interpreter
                        to create the new environment.  The default is the
                        interpreter that virtualenv was installed with
                        (/usr/bin/python)
  --clear               Clear out the non-root install and start from
                        scratch
  .
  .
  .
```

下面让我们一起看一下该工具的使用。和virtualenv一样，首先需要创建一个虚拟的运行环境。

建立虚拟运行环境

```
$ mkvirtualenv newenv
```

当输入上面命令后，也就和virtualenv一样，建立了一个虚拟的运行环境，而且从一开始就处于激活的状态，但我们没有见到newenv目录。

其实virtualenvwrapper对虚拟运行环境环境作了统一管理，根据上面配置的环境变量WORKON_HOME的路径信息，在其中建立了虚拟运行环境目录。在前面我们将WORKON_HOME设定为$HOME/.virtualenvs，让我们用下面的命令对这一目录进行确认，看一下是否有newenv目录生成。

确认虚拟运行环境

```
$ ls -la $HOME/.virtualenvs
```

退出虚拟运行环境的命令同样是deactivate，进入虚拟运行环境的命令变为workon。

进入虚拟运行环境

```
$ workon newenv
```

当我们想浏览所有既存的虚拟运行环境却忘记了它们的名称时，可以不加任何选项，单单输入workon命令即可。虽然该工具包提供了lsvirtualenv这一专门浏览既存虚拟运行环境的命令，但如果仅仅是浏览环境，不如直接使用workon来得方便。

浏览虚拟运行环境

```
$ workon
```

通过mkvirtualenv建立的虚拟运行环境可用下面的命令删除。

删除虚拟运行环境

```
$ rmvirtualenv newenv
```

virtualenvwrapper的基本操作就介绍到这里。如果你想知道更加详细的信息，可以参考下面的文档：http://www.doughellmann.com/projects/virtualenvwrapper/。

## 01.01.04 使用不同版本的Python

Python的版本现在分成2.X和3.X两个分支并行开发。因此，使用Python开发项目时，同样会遇到使用不同的Python版本进行开发的情况。下面我们就如何在同一个操作系统下，使用不同版本的Python分别开发各自的应用程序这一话题进行阐述。

Ubuntu 11.10上默认安装的Python版本是2.7.2。下面我们来安装以下三个不同版本的Python。

- Python 3.2
- Python 2.6
- Python 2.5

Python 3.X并没有在开发项目中大量使用，我们为了接触一下最新的Python，可以先安装一下。Python在Ubuntu 11.10上的安装方法有很多，具有代表性的安装方法有下面几种：

- 通过Ubuntu官方的apt工具包安装
- 通过PPA（Personal Package Archive）的apt工具包安装
- 通过编译Python源代码安装

通过Ubuntu官方包安装一般来说最可靠也最简单。作者在执笔时，可以通过它来安装Python 3.2和Python 2.6。

通过Ubuntu官方的apt工具包安装

```
$ sudo aptitude install python3.2
$ sudo aptitude install python2.6
```

安装完成之后，可以用下面的命令进行确认 。

Python版本的确认

```
$ python3.2 -V
Python 3.2.2

$ python2.6 -V
Python 2.6.7
```

安装的各种各样Python版本均在/usr/bin目录下以带有版本信息的文件名存放。另外，在安装Python 2.5时，输入命令sudo aptitude install python2.5时，会提示无法安装。这是因为在Ubuntu 11.10中，Python 2.5没有被官方apt包放入推荐安装的软件列表中。

遇到Python 2.5这样的老版本软件难以在官方apt资源中找到的情况，我们可以考虑利用PPA（Personal Package Archives）这一由自愿者编译的apt资源站点。由于PPA是自愿者自发支持的资源站点，在使用中往往会有些不兼容的问题存在，所以在使用PPA的资源包时，需要使用者自己承担责任，这一点需要格外慎重。

接下来我们介绍一下从PPA安装Python 2.5的方法。

从PPA（Personal Package Archieves）安装apt工具包

```
$ sudo aptitude install python-software-properties
$ sudo add-apt-repository ppa:fkrull/deadsnakes
$ sudo aptitude update
$ sudo aptitude install python2.5
```

类似使用apt工具包安装Python的工具虽然简单，但有时不一定能够安装到最新版本。因此，在Python出现重要更新时，我们最好学会从源代码直接编译安装。

既然apt官方网站没有推荐Python 2.5，我们就从源代码直接编译安装Python 2.5。首先从下面的URL地址下载Python 2.5的最新版本。

http://www.python.org/download/releases/2.5.6

从源代码编译安装Python没有特别难的地方，按照下面的顺序，使用make命令即可（下面的步骤还包括了使用wget命令下载源代码的过程）。

从源代码编译安装Python

```
$ wget http://www.python.org/ftp/python/2.5.6/Python-2.5.6.tgz
$ tar -xvzf Python-2.5.6.tgz
$ cd Python-2.5.6
$ LDFLAGS="-L/usr/lib/x86_64-linux-gnu" ./configure
$ make
$ sudo make install
```

安装完成后，可以确认一下Python 2.5是否安装在了/usr/local/bin目录下。与此同时，我们也会发现默认的Python版本变成了Python 2.5.6。这是因为操作系统在搜索命令时，是按照PATH环境变量的顺序依次进行搜索的，/usr/local/bin下的Python会比/usr/bin下的Python优先搜索到，并作为默认的Python版本。

**NOTE**

Python的有些模块和apt安装的某些工具包有着依赖关系，如果没有事先安装这些工具包就使用这些模块的话，会发生import错误。

这时我们就需要将那些工具包重新使用apt安装，尤其是下载源代码编译安装之

后，才能继续使用Python某些模块。下面的表格介绍了一些只有安装了apt对应工具包之后才能使用的Python模块。

| Python的模块 | apt的工具包 |
|---|---|
| zlib | zlib1g-dev |
| sqlite3 | libsqlite3-dev |
| readline | libreadline6-dev |
| gdbm | libgdbm-dev |
| bz2 | libbz2-dev |
| Tkinter | tk-dev |

Python 2.5安装确认

```
#环境变量PATH搜索顺序从左往右开始。
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games

# /usr/local/bin/python的执行结果。
$ python -V
Python 2.5.6

# 使用which命令查看实际的路径。
$ which python
/usr/local/bin/python

# 指定路径的话，会发现执行的是Python 2.7.2+。
$ /usr/bin/python -V
Python 2.7.2+
```

### ◉ 使用不同版本Python的方法

最简单的方法是使用virtualenv，建立虚拟运行环境，并在命令行中指出Python的版本信息。

virtualenv执行时指定Python版本

```
$ mkvirtualenv --python=/usr/bin/python2.7 newenv2
```

将想要使用的Python的路径信息放入上面命令，就可以指定在虚拟环境中使用的Python版本。

确认虚拟环境的Python版本

```
(newenv2)$ python -V
Python 2.7.2+
```

如果无法使用virtualenv，可以使用诸如python2.7这类带有版本信息的Python命令，也可以借助软连接将Python命令重定向到某个特定版本。

另外需要注意的是，如果使用源代码编译安装，/usr/local/bin下面的python和

python-config会变成默认的Python。

## 01.02　Mercurial环境搭建

在如今的开发过程中，一般都会引入版本控制系统，这一点不限于Python。一般而言，对于源代码的版本控制常用于多人团队开发，但在管理个人源代码时，同样能够从版本控制系统中受益。比如有一天你突然发现自己开发的应用程序无法工作，就可以通过版本控制系统立刻回退到正确的版本状态；再比如在出现问题时，可以通过版本控制中的时间序列信息，迅速发现问题发生的时间点。

在版本控制系统中，与CVS和Subversion这类集中式的代码库（repository）版本控制系统相比，越来越多的团队开始使用诸如Git和Mercurial这类分布式版本控制系统DVCS（Distributed Version Control System）。Mercurial这类分布式版本控制系统最大的特点是：一个个分散的代码库可以独立使用，可以对其任意进行分支建立、提交、变更、取消等操作，且不会给其他代码库带来任何影响。一个个分散的代码库可视为整个代码库的备份。

### 01.02.01　Mercurial的简介

Mercurial是Linux内核开发人员马特·马考尔（Matt Mackall）于2005年开发的DVCS，与Linux内核之父林纳斯·托瓦兹（Linus Torvalds）开发的Git处于同一时代[①]。现在，Git和Mercurial已经作为DVCS的代表，在业内被广泛地使用。

Mercurial主要使用Python编写[②]，还可以用pip命令进行安装或用Python程序对其功能进行定制。对于已经熟悉Python的用户而言，它非常容易上手；对于使用Python进行开发的团队而言，它非常有亲和力。

Mercurial拥有基于Web的构建管理工具、Bitbucket（https://bitbucket.org/）等知名在线代码托管库、如TortoizeHg（http://tortoisehg.bitbucket.org/）那样的图形用户界面客户端等，这些都是Mercurial的魅力所在。

除上面提到的优点外，Mercurial还有类似svn的命令体系，可以使长期使用svn的用户更容易平滑地迁徙到Mercurial上来，而这一点是Git所不及的。

### 01.02.02　Mercurial的安装

仅通过上面的的说明，你可能无法了解到DVCS的优点，接下来我们将实际操作一下Mercurial的整个安装过程。如果你想了解Mercurial的高级应用，可以阅读本书的第8章。如果你已熟悉Mercurial，可以跳过下面的内容。

① 仅仅比Git晚了不到两周的时间。——译者注

② 有少量C语言编写的代码。——译者注

Mercurial的安装方法和其他基于Linux包的安装方法基本相同，可以使用aptitude、pip以及源代码直接编译等多种方法。本书尽可能介绍最新版的Mercurial的安装，因此使用pip方法来进行。

Mercurial的安装

```
$ sudo pip install mercurial
```

当Mercurial安装完毕之后，就可以使用hg命令了[①]。我们可以输入hg--version来确认Mercurial的版本信息。[②]

Mercurial的安装确认

```
$ hg --version
Mercurial Distributed SCM (version 2.0.1)
(see http://mercurial.selenic.com for more information)

Copyright (C) 2005-2011 Matt Mackall and others
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

## 01.02.03 代码库的建立

在建立Mercurial的代码库之前，需要准备一份关于Mercurial环境配置的清单文件。该文件存放在用户的home目录下，扩展名为.hgrc。该文件的内容如下面例子所示。其中被中括号[ ]括起来的部分称为节（section），如[ui]节就是用来配置Mercurial的用户信息的。

环境配置清单文件中设定的username信息将被作为代码库的账户，在进行提交操作时被Mercurial记录下来，所以请尽可能使用自己的账户名和邮件地址填写环境配置清单文件。

.hgrc的用户名与邮件地址的设置

```
[ui]
username=bpbook <bpbook@beproud.jp>

[extensions]
graphlog=
color=
pager=

[pager]
pager=LESS='FSRX' less
```

[extensions]节用于Mercurial的外部工具扩展。对其中的每一个项目，如果填上外部扩展工具名，就可以在Mercurial使用。不过有些外部工具还需要进行其他设置。如

① mercurial英语中有水银的意思，因此，其命令缩写便使用水银的化学元素符号hg。——译者注

② 本书引用的环境是ubuntu的环境。——译者注

pager，用户在配置了[extensions]节中的[pager]项后，还需要对具体的pager项进行配置，此时[pager]节内容才会正式生效。

当完成了Mercurial环境配置清单文件之后，就可以生成代码库了。如下面的命令所示，在建立了代码库对应的目录之后进入该目录，使用hg init命令。

hg init（初始化代码库）

```
$ mkdir ~/hgtest
$ cd ~/hgtest
$ hg init
```

## 01.02.04　文件的操作

在建立了代码库以后，接下来我们试着往代码库添加一个文件。先建立一个测试文件，接着确认该文件的状态。

代码库的状态确认需要使用hg status命令。如果觉得每次都使用status命令比较麻烦，可以使用hg st缩写命令。hg命令事先均提供了缩写形式，方便用户使用，具体可以参考相关的帮助手册。

hg status（状态的确认）

```
$ touch test.txt
$ hg status

? test.txt
```

在文件名的左侧，记录了该文件在代码库中的状态。在上面例子中，test.txt文件的左侧有一个“？”标记，表示该文件尚未进入Mercurial的管理。接下来，我们将该文件加入到Mercurial的文件管理中去。向Mercurial中添加文件，可以使用命令hg add。

hg add（添加文件）

```
$ hg add test.txt
$ hg status

A test.txt
```

在使用了命令hg add之后，再一次确认文件状态信息，这时可以发现文件前面的标记变为了A，这就表示已经成功地向版本控制软件中添加了文件test.txt。

为了使得所添加的文件在代码库中生效，就需要使用提交（commit）操作。如下例输入命令hg commit。如果没有输入hg commit，所添加的文件并没有真正存在于代码库中，而仅仅是存在于用户对文件操作的地方而已。

hg commit（提交操作）

```
$ hg commit

test commit

HG: Enter commit message.  Lines beginning with 'HG:' are removed.
HG: Leave message empty to abort commit.
HG: --
HG: user: bpbook <bpbook@beproud.jp>
HG: branch 'default'
HG: added test
```

输入hg commit命令以后，文本编辑器会打开，并提示输入注释。这里我们输入test commit。

在注释栏的下面，会看到[branch'default']字样。这是告知用户目前正在操作的开发分支是default分支（即代码库刚刚建立时就已创建的分支）。

NOTE

用户可以修改在提交时输入注释信息的默认编辑器。修改方法如下所示，在原来的.hgrc文件的[ui]节添加一个叫做editor的参数配置即可。

```
[ui]
editor = vim
```

这就表示以后均采用vim作为提交注释的默认编辑器。默认的编辑器同环境变量EDITOR有关，我们可以通过下面的命令来确认默认编辑器是什么。

```
$ echo $EDITOR
vim
```

再一次提交文件之后，使用hg status确认一下内容，会发现和下面给出的结果一样，没有任何输出。

提交确认状态

```
$ hg status
```

接下来我们介绍一下，通过Mercurial如何对编辑过的文档进行操作。

我们用编辑器打开test.txt文件，任意编辑一下，然后保存退出。接着输入hg status查看，会发现文档的状态前面出现了M的标记。状态M表示该文件在提交之后进行了更改，即Modification（变更）。我们可以用hg diff命令，看一下变更前的文件和当前修改过的文件的内容差异。

用hg diff查看内容差异

```
$ hg status
M test.txt

$ hg diff

diff -r 74471564b074 test.txt
--- a/test.txt      Mon Oct 31 18:07:15 2011 +0900
+++ b/test.txt      Mon Oct 31 18:10:45 2011 +0900
@@ -0,0 +1,1 @@
+modify this file
```

在提交前，我们也可以用hg revert命令取消编辑过的文件。

用hg revert取消编辑

```
$ hg revert test.txt
```

如果需要取消已经提交的内容，可以使用hg rollback这个命令，但该命令只能取消前一次提交的内容。

用rollback取消提交

```
$ hg rollback
$ hg status

A test.txt
```

在上面的例子中，我们使用hg rollback命令取消了前一次提交的内容，因此在test.txt文件前面再一次看到了状态A的内容。

接着我们来了解一下如何查看提交的历史信息。我们可以再一次使用hg commit命令提交test.txt，然后输入hg log来查看提交历史。

用hg log查看提交的历史日志

```
$ hg log

changeset:   0:74471564b074
tag:         tip
user:        bpbook <bpbook@beproud.jp>
date:        Mon Oct 31 18:07:15 2011 +0900
summary:     test commit
```

至此为止，我们将Mercurial的基本操作介绍完毕，接下来与此相关的还有如下几个话题：

- 分支的操作
- 使用远程代码库
- 团队开发Mercurial的使用方法

上面的这些话题我们将会在第3章和第8章中详细讨论。

## 01.03 编辑器与方便的开发工具

本节介绍编写Python代码的编辑器、Python的工具模块以及一些开发方面的建议。

### 01.03.01 编辑器

编写Python代码需要一个好的编辑器，最具代表性的编辑器有下面两类：

- Vim（http://www.vim.org）
- Emacs（http://www.gnu.org/software/emacs）

Vim也好，Emacs也好，都是类Unix操作系统下的开发人员非常熟悉的编辑器。这两款编辑器采用了截然不同的操作方式[①]，它们并非Python专用，我周围的开发人员几乎都在使用其中的一款。另外，能够自由更改设置，也是这两款编辑器长盛不衰的原因之一。下面分别介绍一下两款编辑器用来写Python时常用的设置。

#### ◉ Vim

Vim的基本设置是在用户的Home目录下建立一个.vimrc配置文件，Vim根据该文件的设定，体现具体的定制内容。首先根据Python的代码特征，设定其中的换行缩进。打开.vimrc，写下如下内容。

**用于Python的配置**

```
syntax on
filetype plugin indent on
```

- **syntax on（根据Python语法着色）**

  syntax on不仅仅用在Python，在其他编程语言中也会使用，根据语言语法进行高亮着色。

- **filetype plugin indent on（符合Python结构的配置）**

  当打开扩展名为.py的文件，遇到def或者if等Python关键字后再遇到换行时，filetype plugin indent on会将下一行的光标自动缩进到合适的位置。另外，Vim还可以将其配置分散到多个文件中。我们可以根据下面的范例，将所有Vim准备配置Python的内容写入一个文件。

**准备Python配置的文件**

```
$ mkdir ~/.vim
$ mkdir ~/.vim/ftplugin
$ touch ~/.vim/ftplugin/python.vim
```

① 指用户使用编辑器的操作方式。VIM分为三个操作模式，而Emacs则是通过各种组合键完成编辑功能。——译者注

新增一个Python的配置文件python.vim。

**将Tab键换成四个空格**

```
setl expandtab
setl tabstop=4
setl shiftwidth=4
setl softtabstop=4
```

上面的设定完成了以下功能。

- 将所有的Tab键替换成空格键
- 按下Tab键后，自动替换成四个空格键

上面的规则是Python社区推荐的代码风格，叫做PEP8（后面会具体讲到，http://www.python.org/dev/peps/pep-0008），即将缩进用四个空格代替。

下面的设定是在文件保存时，删除每一行末尾的空格。

**删除行末空格**

```
autocmd BufWritePre * :%s/\s\+$//ge
```

PEP8还规定每一行不得超过79个字符，因此到第80个字符时需要自动换行。下面我们就来设置一下。

**每行到第80个字符时自动换行**

```
setlocal textwidth=80
```

如此一来，用Vim编写Python代码的准备工作就完成了。

### ◉ Emacs

Emacs的相关配置按照下面过程按部就班即可。

Emacs在一开始就已经安装了所谓的Python Mode。因此，用Emacs来写Python之前，我们无需做什么特别的设置，最多增加一项将Tab转换为空格就足够了。

配置Emacs是通过.emacs文件来完成的，首先建立该文件。

**创建.emacs文件**

```
$ touch ~/.emacs
```

文件创建好之后，在文件中添加下面的内容。

**将默认缩进改成空格**

```
(setq indent-tabs-mode nil)
(global-font-lock-mode 1)
```

global-font-lock-mode的作用是根据Python语法结构进行高亮区别。

NOTE

python.org的官方wiki里给出了几种编辑器的推荐配置，其中也包含了Emacs的配置。

- Python Editor（http://wiki.python.org/moin/PythonEditors）
- Emacs Editor（http://wiki.python.org/moin/EmacsEditor）
- Emacs Python Mode（http://wiki.python.org/moin/EmacsPythonMode）

### ◉ PyCharm

对于以往使用Eclipse之类的IDE（Integrated Development Environment，集成开发环境）开发的人来说，有时与其使用普通编辑器，不如用IDE来得方便。Python相关的IDE也有几个选择，最近经常听到的是PyCharm（http://www.jetbrains.com/pycharm）。

PyCharm是在2010年发布的收费Python IDE，支持Django开发框架。虽然说是付费软件，但有30天的免费试用期，且同时支持Windows、Mac、Linux等多个操作系统。

对于不太喜欢使用Vim或者Emacs的开发人员来说，可以考虑一下它。

## 01.03.02 Python开发小贴士

本节将会介绍Python开发时，需要知道的一些模式以及Python的常用模块。

### ◉ 交互模式

如果仅仅运行python命令，就会进入Python的交互模式。交互模式是用类似对话方式书写Python代码并执行的一种形式。下面让我们试着直接输入python命令看看。

**启动交互模式**

```
$ python
Python 2.7.2+ (default, Oct  4 2011, 20:06:09)
[GCC 4.6.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

此时进入等待输入的模式，在这个模式下，直接输入Python的代码并且执行，输出会立刻展现在你面前。

**Python代码执行示例**

```
>>> import sys
>>> sys.path
['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-linux2',
'/usr/lib/python2.7/lib-tk', '/usr/lib/python2.7/lib-old',
'/usr/lib/python2.7/lib-dynload', '/usr/local/lib/python2.7/dist-packages',
```

```
'/usr/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages/gtk-
2.0']
>>>
```

这里的sys是Python的基本模块，导入后在屏幕上输出的是Python搜索包路径的所有信息。

执行的结果会保存在_变量中，再一次直接运行它，会得到同样的结果[①]。这种模式对于调试以及验证简单代码都非常方便。退出交互模式的方法是输入Ctrl+d或输入exit()。

**NOTE**

启动Python的交互模式时，会自动加载环境变量PYTHONSTARTUP中配置的文件，如果需要在启动时导入（import）一些文件，可以提前设置好。

另外，在Python的交互模式中，为了方便调试，可以利用一个叫做IPython的小工具，该工具同样可以使用pip安装。

```
$ pip install IPython
```

IPython安装完成后，再一次进入交互模式，就可以完成下面这些之前不能做的事情：

- 使用Tab键补全内容
- 使用常用的Shell命令
- 使用pdb工具调试

### ◉ pep8（检查代码风格的工具）

Python有一个叫做PEP（Python Enhancement Proposals）的社区，该社区的宗旨是进一步完善Python。在社区中常常会给出一些指导性文件。

PEP网址 http://www.python.org/dev/peps/

该社区发布了一个对于Python代码风格的建议性文件PEP8（http://www.python.org/dev/peps/pep-0008）。与之对应，Python提供了一个pep8模块，检查代码是否符合pep8规范。该模块可以通过下面的pip命令安装。

**安装pep8模块**

```
$ pip insatall pep8
```

### ◉ pyflakes（Python语法检查工具）

pyflackes是用来检查Python代码语法销误的模块，它除了检查例行的语法错误外，对于导入的模块却设有最终使用等非语法问题也能及时予以发现。

① 这个特殊变量_和Perl的@_很像，熟悉Perl的读者可以类比。——译者注

pyflakes的使用方法与pep8基本类似，pyflakes命令安装好后，将需要检查文件的文件名作为参数放在该命令后即可。这样该工具就会对该文件做出检查。pyflakes同样可以使用pip安装。

安装pyflakes

```
$ pip install pyflakes
```

NOTE

这里我们介绍了最常使用的pyflakes，除此之外，还有PyLint以及pyChecker等常用工具，你可以根据自身的开发情况来选择。

### ◉ pdb（调试工具）

在最后，我们介绍一下Python的调试工具。对于熟悉C语言调试的开发人员而言，最熟悉的调试工具莫过于gdb；而对于使用IDE的开发人员而言，往往对IDE提供的插入断点、单步执行等功能倍感亲切。

pdb是Python的标准模块配置，因此不需要再安装。最简单的使用方式莫过于在希望代码停下来的位置插入如下代码。

插入pdb代码

```
def add(x, y):
    return x + y

x = 0
import pdb; pdb.set_trace()
x = add(1, 2)
```

执行上面的代码会发现，一旦执行到插入pdp代码的位置，程序就会立刻停了下来，并且还会出现pdb交互方式的界面。

pdb执行的例子

```
$ python pdbtest.py
> pdbtest.py(7)<module>()
-> x = add(1, 2)
(Pdb)
```

本节介绍了有助于Python开发的相关工具、模块等，因为内容比较广泛，所以在介绍具体使用时并未深入。如果你想了解更加详细的内容，可以参考各工具、模块对应的文档。

## 01.04　本章小结

本章围绕着Python个人开发所做的准备，介绍了以下几个内容：

- Python的环境搭建
- Mercurial的环境搭建
- 编辑器以及调试器等工具

介绍了在做开发准备时，如何安装各种各样的工具等内容。这里介绍的方法对于Ubuntu之外的操作系统也基本适用。我们还准备了一些自动的shell脚本、将本书介绍的步骤简化执行等内容，能做的事情还有很多，希望大家一定要下工夫，建立方便使用的开发环境。

NOTE

在进入第2章之前，先来看一下“The Zen of Python”（Python之禅）的相关内容。“The Zen of Python”是Python开发者Tim Peter的文章，他为Python总结出19条特点。这篇文章是在PEP20的时候向开发者公开的（http://www.python.org/dev/peps/pep-0020），在Python的交互模式下，输入import this即可见其具体内容，大家可以确认一下。

```
>>> import this
```

关于这篇文章，大家还可以参考@atsuoishimoto的博客，博客里有关于其内容的详细解说。

- Then Zen of Python 阐释——前篇（http://d.hatena.ne.jp/atsuoishimoto/20100920/1284986066）
- Then Zen of Python 阐释——后篇（http://d.hatena.ne.jp/atsuoishimoto/20100926/1285508015）

# 02 开发Web应用程序

在这一章，我们首先介绍什么是Web应用程序，随后从一个简单的Web留言本范例入手，介绍Web应用程序的基本开发流程。在介绍上述内容的过程中，我们直接使用Python、HTML/CSS的代码。为突出重点，本书不对这些语法做深入介绍。另外，范例的开发环境使用VirtualBox的Ubuntu操作系统、apt安装工具中的Python 2.7，以及virtualenv、virtualenvwrapper。

## 02.01 了解Web应用程序

要开发一个Web应用程序，首先要知道什么是Web应用程序，以及Web应用程序的运行机制。以下，我们就先介绍一下什么样的应用程序才称得上是Web应用程序。

### 02.01.01 什么是Web应用程序

Web应用程序之所以以Web开头，是因为所有的Web应用程序都必须经过Internet才能使用[①]。例如，Google搜索服务、Gmail、Wikipedia、各种博客服务、Twitter等微博服务、GREE等移动游戏、Facebook等SNS网站以及Facebook中所有的社交类游戏等，都可以称作Web应用程序[②]。当用户使用Web应用程序的时候，需要打开Web浏览器才能连接并使用应用程序。

如此，就会产生一些问题：使用Web浏览器浏览的Web站点是什么呢？只展示一些图片和文字的Web站点能不能叫做Web应用程序呢？

答案是，不能一概而论。因为它们也有可能是Web应用程序。表面上看似简单的Web站点，可能是由CMS（Content Management System，内容管理系统）等Web应用程序所构建的。Web浏览器所做的是连接到目标服务器，从服务器上将HTML、CSS以及图片等下载下来，展示在浏览器中。如果Web站点的服务器仅仅配置了一些单一不变的内容供浏览器下载并展示，这样的Web站点就不能称为Web应用程序。只有那些根据用户的输入内容，动态生成相关页面并返回给浏览器的系统才能称作Web应用程序。[③]

---

① 这里的Internet是指用Internet的相关协议。——译者注

② 目前Facebook和Twitter都没有进入中国，国内类似的有人人和新浪微博。——译者注

③ 即通常说的动态Web站点和静态Web站点，动态Web站点才称得上Web应用程序。——译者注

NOTE

CMS是管理Web站点中的文章、图片、通信等的系统。Wiki以及博客系统都属于CMS。

## 02.01.02　Web应用程序与桌面应用程序的不同

与使用Web浏览器进行访问的Web应用程序相对应的，是那些需要安装在计算机上才能使用的桌面应用程序。下面这张表格展现了Web应用程序和桌面应用程序的不同。

| 比较项目 | Web应用程序 | 桌面应用程序 |
| --- | --- | --- |
| 服务器 | 必须 | 可有可无 |
| Internet连接 | 必须 | 根据应用程序而定 |
| 使用操作系统的特性 | 不可能 | 可能 |
| 安装 | 不需要 | 必须 |
| 版本升级 | 用户可以不关心 | 用户执行 |
| 跨平台使用 | 只要有浏览器就可以运行 | 需要针对不同平台 |
| 移动终端使用 | 和PC使用类似 | 需要针对不同平台 |
| 运行速度 | 慢 | 快 |

根据上面这张表的内容，可见Web应用程序和桌面应用程序各有优缺点。在实现通过网络获取信息的功能时，二者都是需要服务器的，而使用Web应用程序实现起来相对简单一点。随着时间的推移，表中的内容也在发生着变化。

例如，以往的Web应用程序需要连接Internet才能使用，如今出现了脱机离线也可以使用的机制。另外，桌面应用程序的版本升级也出现了类似Google Chrome这样自动升级的例子，用户无需费心手动升级。Web应用程序和桌面应用程序的距离正在一天天缩短。

## 02.01.03　Web应用程序的机制

接下来我们就Web应用程序的机制做一个说明。

① 用户通过Web浏览器输入URL信息。

② Web浏览器根据输入的URL域名信息向DNS服务器查询该域名对应的IP地址。

③ Web浏览器获得IP地址后，连接该地址的Web服务器，通过HTTP协议进行通信。

④ Web服务器通过HTTP协议得到信息，执行对应的Web应用程序并获得执行的内容。

⑤ Web服务器将执行Web应用程序得到的HTML、CSS以及JavaScript、图像等内容返回给浏览器。

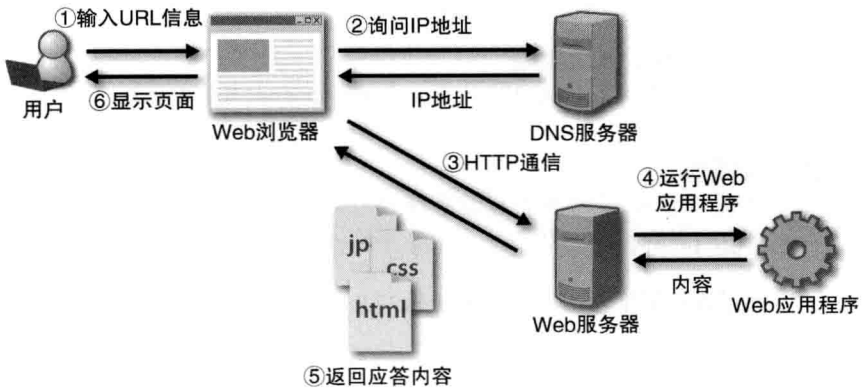
⑥ Web浏览器根据Web服务器得到的信息用页面的形式展现给用户。

就上面流程中提到的术语，读者可以查阅下面这张表。

| | |
|---|---|
| Web浏览器 | 通过HTTP协议等与目标计算机进行通信，将获得的HTML、CSS、图片等内容信息通过页面展示给用户的软件 |
| URL | Uniform Resource Locator的缩写。包含域名信息，用来表示Internet上可访问内容的资源位置的文字序列 |
| 域名 | 与IP地址相关联的文字序列（如URL中含有的www.beproud.jp这样的文字） |
| DNS服务器 | 提供通过域名获得的对应IP地址服务的服务器 |
| IP地址 | 通信对端计算机在网络上的地址。（以IPv4地址为例，一般表示为192.168.0.1这样用0~255之间的数值串联起来的、通过点进行分割的数字序列） |
| HTTP | Hypertext Transfer Protocol缩写，同目标计算机进行通信的协议 |
| HTML | 一种用来描述文档结构的语言，所描述的文档可以包含文本、图像等信息 |
| CSS | 描述HTML等中记录的内容该如何表示、展现的语言或者语法组合 |
| Javasccript | 在浏览器上执行的程序 |
| Web服务器 | 通过HTTP协议进行通信的服务器或者计算机 |
| Web应用程序 | 在Web服务器上执行的程序 |
| CGI | Common Gateway Interface的缩写，Web应用程序构成的一种方式 |

上述的流程可以用图2-1表示。

图2-1 Web应用程序的运行流程




## ◉ Web应用程序与CGI

CGI是Web服务器执行Web应用程序的一种形式。Web服务器通过执行CGI程序（CGI脚本），将CGI程序的标准输出的内容直接通过HTTP协议返回。最简单的CGI程序是通过控制台终端仅仅显示的一串文字而已。在Python的标准模块工具中，有

一个叫做CGIHTTPServer的CGI模块可作为Web服务器运行。下面我们就来实际操作一下。

help.py

```
#!/usr/bin/env python
print "200 OK"
print "Content-Type: text/plain"
print ""
print "Hello CGI!"
```

上面这个脚本用Python直接运行，就会在控制台屏幕上输出四行文字。

如果用CGIHTTPServer执行CGI的话，需要将该文件放入一个叫做cgi-bin的目录下，我们可以建立该目录，将hello.py放入其中。文件放到指定位置后，通过python命令使用-m选项执行CGIHTTPServer模块。

```
$ mkdir cgi-bin
$ mv hello.py  cgi-bin/
$ python -m CGIHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

该命令执行后，在执行的环境中（这里是VirtualBox上的Ubuntu）就会打开8000的端口进行侦听，等待请求的到来，如此一来Web服务器便正式启动了。停止Web服务器的方法是按下Ctrl+C键。

本地环境（即Host的操作系统）中的Web浏览器如果需要访问上面的服务，必须通过SSH连接建立一个隧道。如果需要建立一个本地的8000端口和VirtualBox上的8000端口的隧道，可以使用下面的命令。

通过SSH建立8000端口的隧道

```
$ ssh -p 2222 -L 8000:127.0.0.1:8000 bpbook@127.0.0.1
```

NOTE

*上面的命令假设客户机操作系统上存在bpbook用户，且2222端口已经绑定SSH。SSH的绑定等操作读者可以参考附录B。*

通过Web浏览器访问地址http://127.0.0.1:8000/cgi-bin/hello.py，就可以看到浏览器中显示Hello CGI!这样的信息。

NOTE

*127.0.0.1是表示计算机本身的环回地址。*

这个CGI程序的例子是使用Python编写的，如果单从CGI的输出来看，和具体编程语言的实现是无关的。

◉ Web应用程序和应用服务器

应用服务器起到了执行Web应用程序，并同Web服务器进行通信的作用。在整个执行过程中，它通过等待来自Web服务器的请求，将该请求再通过Web应用程序执行并返回。整个程序一般都是直接加载在内存中执行的，因此比普通的CGI要快很多。Web服务器和应用服务器之间的通信一般使用HTTP或者FCGI协议进行。使用HTTP协议进行通信的应用服务器可使用Web浏览器直接访问。

本章使用的Flask已经内置了一个供开发使用的应用服务器（即Web服务器），没有必要再准备其他的Web服务器。

## 02.02 Web应用程序开发的事前准备

这里首先介绍一下Web应用程序开发的事前准备。前提是使用Python与virtualenv、virtualenvwrapper，关于这些环境的建立可以参考第1章。

### 02.02.01 关于Flask

本章要做的留言簿Web应用程序采用一种叫做Flask（http://flask.pocoo.org/）的Web应用程序框架，需要事先安装。Flask基于Werkzeug（WSGI Utility）和Jinja2（Template Engine）两个Web应用库，是Python的Web应用开发框架的一种。对于一些小的Web应用程序可以非常快速地完成。

### 02.02.02 Flask的安装

安装Flask之前，首先用mkvirtualenv命令建立一个供开发应用时运行的虚拟运行环境。建立一个名叫guestbook的虚拟运行环境可以用下面的命令。

**建立一个名为guestbook的虚拟运行环境**

```
$ mkvirtualenv guestbook
```

建立完成虚拟运行环境后，可以使用pip命令安装最新版本的Flask。在guestbook虚拟运行环境有效的状态下，使用以下命令。

**使用pip命令安装Flask**

```
$ pip install -U flask
```

这里指定-U参数，表示如果已经安装了旧版本的Flask的话，用新版本去替换旧版本。到本书执笔时为止，使用的是Flask 0.8。至此Flask的安装已经完成，准备也到此结束。

## 02.03　Web应用程序的开发流程

接下来Web应用程序的开发该如何进行才好呢?

我们只是决定做一个留言簿的Web应用，首先需要确定该应用有哪些需求。从确定需求开始，考虑如何设计画面和功能，然后考虑如何实现编码，最后确认Web应用程序的各项功能。

上述顺序经过整理后如下所示：

① 从需求明确必须具备的各项功能
② 明确从各项功能开始的页面表现形式
③ 页面设计
④ 功能制定
⑤ 页面和对应功能相组合
⑥ 确定功能的运行
⑦ 完成

下面我们从上面的每一项入手，一边开发目标Web应用程序，一边进行说明。

## 02.04　明确需要开发的目标系统

以下将对Web应用程序涉及的需求与必要功能、页面进行说明。

### 02.04.01　留言簿应用程序的需求

我们首先确定在留言簿应用中需要做些什么（需求）。

如果在一开始没有明确需求与样式，那么在开发过程中就会产生很多疑惑。例如不知道哪些功能是必须的，哪些功能可有可无等。

在这里，我们明确Web留言簿应用会有下面这些需求：

① 提供用户通过浏览器发表评论的表单
② 发表评论的表单中可以输入用户名和评论内容
③ 评论表单提交后保存评论者的姓名和评论内容
④ 评论内容、评论者姓名、评论时间需要体现在页面中
⑤ 整个应用由一个页面组成，上面是发表评论的表单，下面显示评论内容
⑥ 最新的评论内容总是出现在页面的最顶端
⑦ 通过互联网使用该留言簿
⑧ 可以显示多台计算机发表的评论内容

### 02.04.02 明确必要的功能

当明确各个需求之后，就需要考虑为满足那些需求要实现哪些功能。

这次开发的是Web应用程序，需要使用Web应用程序的开发框架。因此，上面需求中的①、⑦、⑧这3条可以不用深入考虑。

根据要开发的Web应用程序的需求考虑要实现的功能，可以参考下表。

| 功　能 | 说　明 | 需求编号 |
|---|---|---|
| 发表评论 | 提供可以输入姓名、评论内容的表单并在提交过程中保存数据 | ①、②、③ |
| 显示评论 | 提取评论发表中保存的数据（按评论日期排列），将取出的信息显示在页面中 | ④、⑥ |
| Web应用程序 | 用Web浏览器显示，通过互联网可以让多台计算机同时使用 | ①、⑦、⑧ |

在各个需求中，⑤是页面相关的内容，会在接下来的小节中进行说明。这里我们可以确认一下，是否该有的功能都已经出现在了上面的表格中。

### 02.04.03 明确页面的表现形式

从Web应用程序需要实现的功能入手，考虑需要什么样的页面与之配合，对应的内容可以参见下面这张表格：

| 功　能 | 必要的页面要素 | 说　明 |
|---|---|---|
| 评论发表 | 姓名与评论的表单 | 姓名与评论的输入框和提交按钮 |
| 评论展示 | 姓名与评论的显示 | 发表评论人与评论页面的列表显示 |

由于存在所有功能需要在一个页面内表示的需求（需求⑤），因此需要将上面两个功能在一个页面里面体现，因此页面形式要求如下所示：

| 必须的页面内容 | 页面要素 |
|---|---|
| 评论投稿、页面表示 | 姓名与评论的表单、姓名与评论的列表显示 |

## 02.05 页面设计

明确了功能和页面的要求之后，就可以设计页面了。

页面设计要先于功能的实现是因为页面设计工作所完成的成果更容易给人一种应用程序完成的整体印象。在系统功能同页面整合前，页面设计工作只需给出所需的HTML和CSS文件即可。

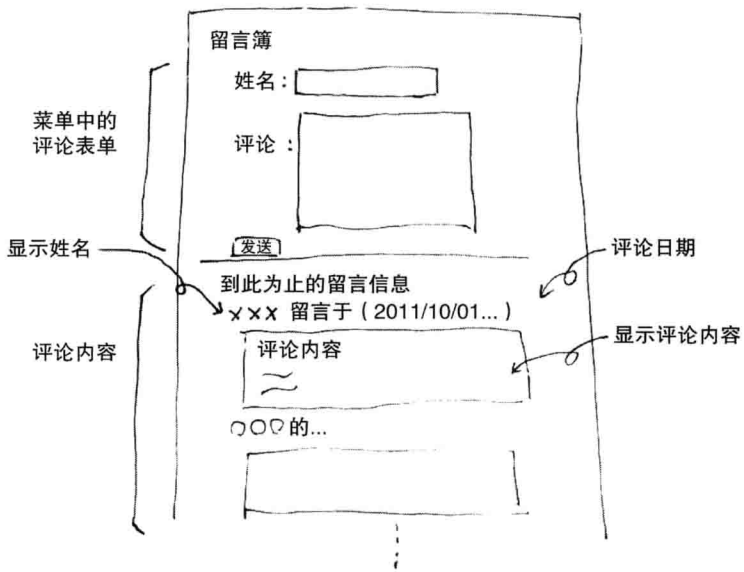
## 02.05.01　页面设计草图

在开始编写HTML代码之前，依然有必要明确我们需要设计的是一个怎样的页面，将想法画在草图上，叫做“设计草图”。

最初阶段我们推荐在纸上画草图。之后如果想修改，或需要使该草图显得漂亮美观的话，也可以使用各种图像处理软件修饰。

这里，我们不使用任何工具，如图2-2所示，直接画一张草图。

图2-2　留言簿Web应用程序的页面草图




上面这张草图的设计和需求的表单一致：上半部分是表单，下半部分显示提交的评论内容。在这张草图中一些要素（表单或者评论内容）的注释也写在了旁边。有了这样的草图，既可以为之后功能与页面的组合提供参考，还能够发现功能上遗漏的地方，非常有用。

## 02.05.02　设计HTML和CSS

接下来，我们来设计HTML和CSS文件。为了能够在Web浏览器中进行验证，设计工作将在本地环境（Host的操作系统）中进行。最后我们将设计好的文件通过SCP命令上传到客户机服务器上，以备后用。

首先根据草图的要求，编写HTML文件。这里我们并不关心整个页面的布局（layout），因为页面布局将在设计CSS时进行调整。

下面是HTML文件的源代码。

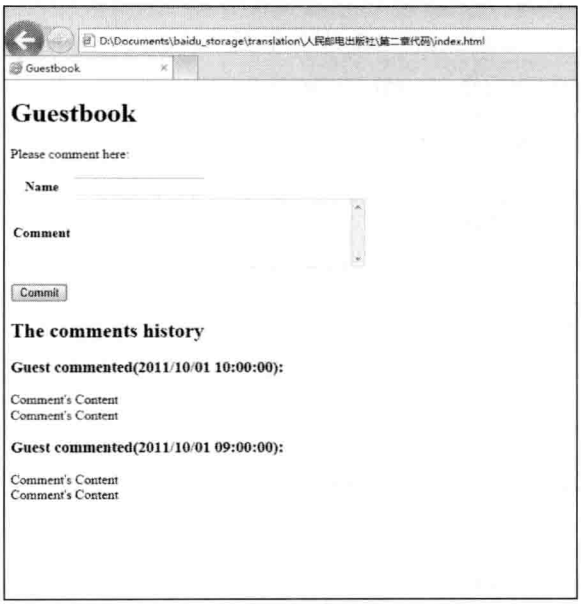
index.html

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 //EN"
  "http://www.w3.org/TR/xhtml11/DTD/ xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <title>Guestbook</title>
  </head>
  <body>
    <h1>Guestbook</h1>
    <p>Please comment here:</p>
    <form action="/post" method="post">
      <table>
        <tr>
          <th>Name</th>
            <td>
              <input type="text" size="20" name="name" />
            </td>
        </tr>
        <tr>
          <th>Comment</th>
          <td>
            <textarea rows="5" cols="40" name="comments"></textarea>
          </td>
        </tr>
      </table>
      <p><button type="submit">Commit</button></p>
    </form>

    <h2>The comments history</h2>
    <h3>Guest wrote comments(2011/10/01 10:00:00):</h3>
    <p>
      Comments<br/>
      Comments
    </p>
    <h3>Guest wrote comments(2011/10/01 10:00:00):</h3>
    <p>
      Comments<br/>
      Comments
    </p>
  </body>
</html>
```

上面的HTML文件用Web浏览器打开后如图2-3所示。

图2-3　只有HTML的页面




HTML文件完成之后，开始编写CSS文件。通过该文件可以调整页面字体的颜色、字号的大小以及显示位置等风格（style）。

下面的就是完成的CSS文件。

main.css

```
body {
  margin:0;
  padding:0;
  color:#000E41;
  background-color:#004080;
}

h1 {
    padding:0 1em;
    color: #FFFFFF;
}

#main {
    padding:0;
}

#form-area {
    padding:0.5em 2em;
    background-color:#78B8F8;
}

#entries-area {
```

```
    padding:0.5em 2em;
    background-color: #FFFFFF;
}

.entry p {
  padding:0.5em 1em;
  background-color: #DBDBFF;
}
```

将CSS文件里面记录的id和class设置在HTML文件的指定位置，如下所示。

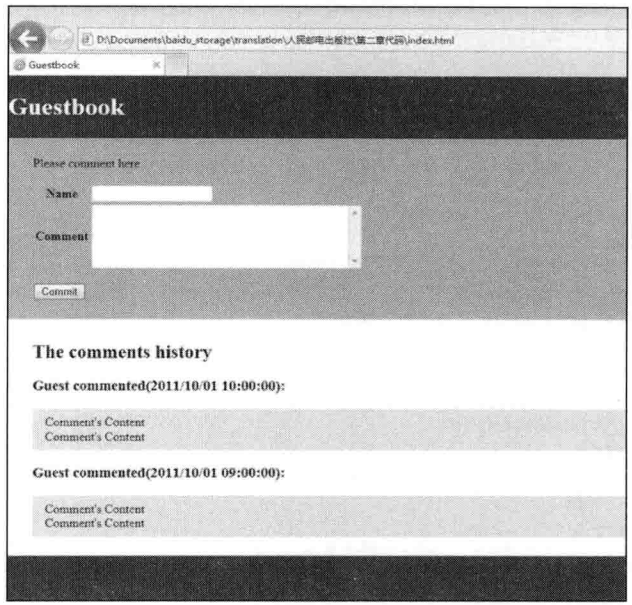
index.html

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 //EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <title>Guestbook</title>
    <link rel="Stylesheet" href="main.css" type="text/css" />
  </head>
  <body>
    <div id="main">
      <h1>Guestbook</h1>
        <div id="form-area">
          <p>Please comment here:</p>
          <form action="/post" method="post">
            <table>
              <tr>
                <th>Name</th>
              <td>
                <input type="text" size="20" name="name" />
              </td>
            </tr>
            <tr>
              <th>Comment</th>
              <td>
                <textarea rows="5" cols="40" name="comments"> </textarea>
              </td>
            </tr>
          </table>
          <p><button type="submit">Commit</button></p>
        </form>
      </div>
      <div id="entries-area">
        <h2>The comments history</h2>
        <div class="entry">
          <h3>Guest commented(2011/10/01 10:00:00):</h3>
          <p>
              Comment's Content<br />
              Comment's Content
          </p>
          <h3>Guest commented(2011/10/01 09:00:00):</h3>
```

```
            <p>
              Comment's Content<br />
              Comment's Content
            </p>
          </div>
        </div>
      </div>
    </body>
</html>
```

适用后的HTML文件通过Web浏览器打开后，就是图2-4的样子了。

图2-4　应用了CSS的页面




到此为止，整个Web留言簿页面部分的HTML和CSS就完成了。将完成后的文件通过scp命令上传到VirtualBox的Ubuntu上，传送的命令如下所示。[①]

通过scp命令上传文件

```
$ scp -P 2222 index.html main.css bpbook@127.0.0.1:~/
```

NOTE

SSH连接的配置等内容请参考本书附录B。

① 本书的Host的机器也是Linux，因此可以直接使用scp，使用Windows读者可以用ftp。——译者注

## 02.06 后台功能的开发



终于到了写Python代码的时候了。这个Web留言簿的必要功能是通过服务器端的程序实现的。我们优先开发重要的功能，以及那些能提高开发效率的功能。

Web留言簿是围绕保存评论数据和提取评论数据这两部分展开的，所以首先从这两个功能入手。

### 02.06.01 保存评论数据

评论功能必须保存表单中的姓名、评论内容以及评论日期。

在这里要保存评论数据，我们使用Python的标准模块shelve。shelve模块是Python字典类型的一种，可以像访问对象一样操纵数据，将对象持久保存。也就是说，我们将评论内容的数据放在字典对象中，将多个字典通过列表的形式放到shelve中去。

接下来我们就开始编写留言簿的脚本文件。其中包含了保存数据函数save_data的实现，整个guestbook.py如下所示：

guestbook.py

```
# coding: utf-8
import shelve

DATA_FILE = 'guestbook.dat'

def save_data(name,comment,create_at):
    """Save the comment data
    """

    # open the shelve module database File

    database = shelve.open(DATA_FILE)
    # if there is no greeting_list in database, create it.
    if 'greeting_list' not in database:
        greeting_list=[]
    else:
        # get the greeting_list from database
        greeting_list=database['greeting_list']

    # append the data into the list top
    greeting_list.insert(0, {
        'name':name,
        'comment':comment,
        'create_at':create_at,
    })

    # update the database
    database['greeting_list']=greeting_list

    # close the database file
    database.close()
```

我们来看一下这个save_data函数是如何运行的。通过终端，我们从放有guestbook.py这个文件的目录中启动Python的shell。通过Python的shell将guestbook模块中的save_data函数载入并运行。

save_data函数的运行与确认

```
$ ls  # confirm the guestbook.py is put under current directory.
guestbook.py
$ python # start Python shell
>>> import datetime
>>> from guestbook import save_data
>>> save_data('test','test comment',
              datetime.datetime(2011,10,1,10,0,0))
>>>
```

create_at的参数是以datetime模块的时间日期对象的形式传入并保存的。如果没有报错，说明该功能已经被执行了。为了进一步确认数据有没有被保存进字典，我们紧接着就编写将保存的数据读出的功能。

## 02.06.02　取出保存过的评论数据

将数据从事先保存进的shelve对象模块中取出，我们可以用一个函数来完成这一功能。在guestbook.py中添加一个load_data函数。

guestbook.py

```
def load_data():
    """Return the comment data saved before
    """
    # open the shelve module database file
    database = shelve.open(DATA_FILE)

    # get the greeting_list. if not ,just return empty list.
    greeting_list = database.get('greeting_list',[])

    database.close()

    return greeting_list
```

接着我们同样可以在Python的shell中，将该函数载入并且执行。

load_data函数运行与确认

```
>>> from guestbook import load_data
>>> load_data()
[{'comment': 'test comment', 'name': 'test', 'create_at': datetime.
datetime(2011, 10, 1, 10, 0)}]
```

如果运行正确，将会得到我们之前保存的数据。

## 02.06.03 使用模板引擎显示页面

为了让从文件中取出的数据显示在页面中，我们需要使用模板引擎（template engine）。模板引擎是将内置的原型模板和填入模板的数据进行组合并且输出的程序。Flask支持标准模板引擎Jinjia2。

为了生成页面所表示的HTML文件，我们可以建立一个文件夹用来存放生成HTML文件的模板。

配置模板文件夹

```
$ mkdir templates
$ mv index.html templates/
```

接下来，服务器程序就会使用该HTML文件（模板），生成最终展现给用户的页面。为了使用Flask的相关功能，往guestbook.py的头部添加表示首页和启动Web服务器的相关代码。

guestbook.py

```
# coding: utf-8

import shelve

from flask import Flask, request, render_template, redirect, escape, Markup

application = Flask(__name__)

DATA_FILE = 'guestbook.dat'

def save_data(name, comment, create_at):
    """Save the comment data
    """
    ...

def load_data():
    """Return the comment data saved before
    """
    ...

@application.route('/')
def index():
    """Top page
    Use template to show the page
    """
    return render_template('index.html')

if __name__ == '__main__':
    # Run the application when the IP address is 127.0.0.1 and the port is 5000

    application.run('127.0.0.1',5000,debug=True)
```

在以上程序中，application变量为Flask类的实例，构造时使用了_name_参数

表示整个程序的模块名。其中route方法是对URL进行处理的装饰器，其作用是表明网站首页地址的URL（/），并且在服务器端执行被修饰的index函数。render_template函数将其输入参数中指定的文件作为模板，通过模板引擎将新内容输出到浏览器[①]。

用Python命令来执行一下guestbook.py。

**执行完成的Web应用程序**

```
$ python guestbook.py
 * Running on http://127.0.0.1:5000/
 * Restarting with reloader
```

执行guestbook.py后，执行的操作系统（这里是VirtualBox上安装的Ubuntu）就以127.0.0.1的5000端口为服务入口，启动Web应用服务（Web服务器），等待请求的到来。如果需要停止该服务进程，按下Ctrl+C即可。

本地环境（Host操作系统）需要通过Web浏览器访问上述页面，因此需要通过SSH建立隧道。在Host操作系统上执行下述命令，就可以建立Host操作系统的5000端口和Guest操作系统的5000端口的隧道[②]。

**为了能够从Host操作系统上访问Guest操作系统的Web服务而执行的命令**

```
$ ssh -p 2222 -L 5000:127.0.0.1:5000 bpbook@127.0.0.1
```

**NOTE**

SSH连接的配置请参考本书附录B。

通过Web浏览器打开http://127.0.0.1:5000的地址，确认页面是否出现。由于出现的页面没有使用CSS文件，我们需要对该页面进行配置。我们可以在Flask下的static目录中建立该文件。首先建立一个static目录，然后将main.css文件放入其中。

**静态文件配置**

```
$ mkdir static
$ mv main.css static/
```

另外，index.html所引用的CSS文件路径需要对应修改为：/static/ main.css：

**templates/index.html**

```
<link rel="Stylesheet" href="/static/main.css" type="text/css" />
```

当再次打开该页面时，我们发现页面已经应用了CSS文件中的样式，接着就可以将从数据库中取出的数据显示在页面上了。使用guestbook.py中的index函数调用load_data函数，将取得的数据放在模板中。

① 这里提到的模板引擎和PHP的模板引擎概念类似。——译者注

② 这条命令需要Host的操作系统上有SSH命令工具。如果读者使用Windows的话，需要有Cygwin等支持。——译者注

guestbook.py

```
@application.route('/')
def index():
    """Top page
    Use template to show the page
    """
    #read the comment data

    greeting_list=load_data()
    return render_template('index.html',greeting_list=greeting_list)
```

render_template函数可以将指定关键字的值作为模板变量使用。在本例中，就是greeting_list作为模板变量的值传递给模板引擎。

在模板文件templates/index.html中，也需要使用模板变量进行对应的修改[①]。HTML文件的修改内容如下所示：

templates/index.html

```
<div id="entries-area">
  <h2>The comments history</h2>
  <div class="entry">
    {% for greeting in greeting_list %}
      <h3>{{greeting.name}}
            commented at {{greeting.create_at}}:</h3>
      <p>{{greeting.comment}}</p>
    {% endfor%}
  </div>
</div>
```

模板内部可以使用一种称为模板语言的特别标识。在Jinja2这个模板引擎中，通过{% …%}这种形式的标签可以使用if以及for等控制语句。模板的变量部分写在{{...}}中。在上面例子中，模板引擎将greeting_list中的值一个接一个地取出后，替换掉模板引擎中的greeting变量，并在for和endfor之间反复进行这类模板引擎变量的替换，从而输出到显示页面上。

这样一来，就可以将save_data保存的数据显示在页面上了。

## 02.06.04 准备评论的目标URL地址

接下来实现将评论的表单内容通过save_data函数进行保存的功能。由于模板文件中表单的action属性使用了/post方法，因而需要将该URL完成[②]。由于还用到了datetime模块，因此要在shelve的下一行加上下面这一行指令。

导入datetime模块

```
from datetime import datetime
```

① 和PHP等类似，在模板文件中定义一些变量，由模板引擎根据实际内容进行替换。——译者注

② 关于POST和GET等表单内容，读者可以参阅HTML相关知识。——译者注

经在Guest OS中安装完毕SSH服务，无需另外安装软件。在Guest OS中的SSH守护进程是否正常启动，可以通过下面命令进行确认。

**进程的确认**

```
$ ps aux |grep sshd
```

如果在进程的一览中出现了“/usr/bin/sshd -D”的信息时，说明SSH的守护进程已经正常启动。

**NOTE**

如果当时忘记安装SSH服务的话，可使用aptitude命令继续安装。

```
$ sudo aptitude install ssh
```

当SSH准备完毕后，便可以从Host OS登录到Guest OS。在VirtualBox中提供了几种连接方式，这里介绍其中的NAT连接方法，即使用端口映射的机制将Host OS上的2222端口连接至Guest OS上的SSH端口（默认是端口22）。

从VirtualBox管理界面中选择“设置”→“网络”→“网卡1”→“端口映射”。如图B-20。

**图B-20　VirtualBox的端口映射配置**




按照上图设置后，在Host OS的控制台下使用下面命令即可连接。

**通过SSH连接**

```
$ ssh -p 2222 bpbook@127.0.0.1
```

NOTE

这里假设Host OS是“Mac OS X”，可以直接使用ssh命令。如果是Windows，则没有ssh命令。这时需要使用类似“PuTTY”这类工具替代，完成SSH连接的工作。

- PuTTY

http://www.chiark.greenend.org.uk./~sgtatham/putty/

“@”之前的bpbook是在安装Ubuntu时创建的用户名，可以替换成合适的用户名。而密码则输入之前创建的内容即可。成功登录系统后可以看见图B-21所表示的画面。

图B-21　通过SSH从Host OS登录Guest OS后的画面

```
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-13-server x86_64)

 * Documentation:  https://help.ubuntu.com/11.10/serverguide/C

  System information as of Mon Dec 19 15:02:41 JST 2011

  System load:  0.0               Processes:           56
  Usage of /:   25.0% of 7.12GB   Users logged in:     0
  Memory usage: 12%               IP address for eth0: 10.0.2.15
  Swap usage:   0%

  Graph this data and manage this system at https://landscape.canonical.com/
Last login: Mon Dec 19 14:43:28 2011 from 10.0.2.2
                    :~$
```

NOTE

一般而言，SSH连接服务器只使用密码口令登录是一种不太安全的方式。本节的内容归根到底也是属于个人开发环境的配置范畴，只对使用密码口令登录的方式进行相关说明。要想让登录环境更加安全，推荐使用“公开密钥”认证的方式。

- 公开密钥

http://zh.wikipedia.org/wiki/%E5%85%AC%E5%BC%80%E5%AF%86%E9%92%A5[①]

## B-03　配置中文

中文本地化配置

```
sudo apt-get install language-pack-zh
sudo locale-gen zh_CN.GBK
sudo dpkg-reconfigure locales
export LANG="zh_CN.UTF-8"
export LANGUAGE="zh_CN:zh:en_US:en"
```

以上命令执行完毕后，中文的locale文件便会安装完成。随后在登录的Shell配置文件中（.bashrc）中添加下面的环境变量。

① 这里是wikipedia的中文站点连接。——译者注

在.bashrc中添加语言配置的环境变量

```
export LANGUAGE=ja_JP:ja
export LANG=ja_JP.UTF-8
```

到此为止，配置便完成了。之后通过启动中文对应的Vim等软件，确认是否已显示中文界面。

[General Information]
书名=Python开发实战
作者=日本BePROUD股份有限公司著
丛书名=图灵程序设计丛书
页数=378
SS号=13532824
出版日期=2014.06
出版社=北京市：人民邮电出版社
ISBN号=978-7-115-32089-6
中图法分类号=TP311.56
原书定价=79.00
参考文献格式=日本BePROUD股份有限公司著.Python开发实战.北京市：人民邮电出版社,2014.06.
内容提要=本书来自真正的开发现场，是BePROUD公司众多极客在真实项目中的经验总结。作者从Python的环境搭建开始讲起，介绍了Web应用开发方法、任务管理，以及评审、测试及高效部署、服务器调试等内容，尽可能网罗了Python开发流程中所涉及的方方面面。在这里，Python仅仅是一个载体，很多知识点在非Python下也适用，这也是本书最大的特色所在。

# 有关此电子图书的说明

本人由于一些便利条件，可以帮您提供各种中文电子图书资料，且质量均为清晰的PDF图片格式，质量要高于网上大量传播的一些超星 PDG 的图书。方便阅读和携带。只要图书不是太新，文学、法律、计算机、人文、经济、医学、工业、学术等方面的图书，我都可以帮您找到电子版本。所以，当你想要看什么图书时，可以联系我。我的 QQ是：85013855，大家可以在 QQ上联系我。

此 PDF 文件为本人亲自制作，请各位爱书之人尊重个人劳动，敬请您不要修改此 PDF文件。因为这些图书都是有版权的，请各位怜惜电子图书资源，不要随意传播，否则，这些资源更难以得到。